МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторної роботи 5
за темою «Робота зі структурами та файлами»
з курсу «Алгоритмізація та програмування. Частина 1»

для студентів спеціальності
122 «Комп’ютерні науки»

Харків 2019

Методичні вказівки до виконання лабораторної роботи 5 за темою «Робота зі структурами та файлами» з курсу «Алгоритмізація та програмування. Частина 1» для студентів спеціальності 122 «Комп'ютерні науки» / уклад. Л. В. Іванов, М. О. Білова. – Харків : НТУ «ХПІ», 2019. – 18 с.

Укладачі: Л. В. Іванов
М. О. Білова

Рецензент О.В. Шматко

Кафедра програмної інженерії та інформаційних технологій управління

ЗМІСТ

Вступ

Курс «Алгоритмізація та програмування» присвячений теоретичним та практичним аспектам розробки алгоритмів і програм мовою С++. Отримані в результаті вивчення даної дисципліни знання та навички можуть бути використані в усіх наступних курсах, під час виконання курсових та дипломних проектів, вивчення дисциплін, що пов'язані з обчислювальною технікою та програмуванням.

У процесі виконання лабораторної роботи за темою «Робота зі структурами та файлами» студенти мають закріпити теоретичний матеріал, отримати навички практичної роботи при реалізації програм, пов'язаних з зчитуванням та записом даних у текстовий файл.

Перша частина стосується використання масивів символів, структур, зв'язних списків. Охарактеризовано особливості і наведено приклади використання введення та виведення у стилі мови програмування С, розглянуто роботу з файлами. У практичній частині подано код для трьох програм: розрахунку суми, відстані між точками, сортування у зворотному порядку зі зчитуванням та записом у новий txt-файл.

Друга частина складається з завдань для лабораторної роботи, що використовуються як поточний контроль засвоєння матеріалу. Лабораторні заняття розраховані на роботу в комп'ютерному класі під безпосереднім керівництвом викладача та самостійну роботу студентів, яка передбачає написання тексту програми, закріплення практичних навичок роботи на комп'ютері, набутих при виконанні відповідної лабораторної роботи. Варіанти індивідуальних завдань видаються викладачем на занятті.

У методичних вказівках подано три завдання. У цілому методичні вказівки будуть корисні студентам різних спеціальностей і форм навчання, які вивчають мову С ++.

ТЕОРЕТИЧНА ЧАСТИНА

1. Масиви символів

У мові C++ (як і в C) рядок – це масив символів, що закінчуються нульовим символом (символ з кодом 0). Можна створити й ініціалізувати рядок так само, як і будь-який інший масив. Наприклад:

```
char name[] = {'A', 'n', 'd', 'r', 'e', 'w', '\0'};
```

Останній символ `'\0'` використовують як ознаку закінчення рядка. Рядок може бути ініціалізований літералом:

```
char name[] = "Andrew";
```

Додавати нульовий символ вручну не треба, оскільки компілятор додає його автоматично. Таким чином, цей масив має сім елементів.

Мова C++ дозволяє читати масив символів без циклу. Важливо переконатися, що є достатньо пам'яті для розташування символів:

```
char s[30];
cin >> s;
```

Вказівники на символ можна ініціалізувати строковими літералами:

```
char *st = "C++";
```

Компілятор виділяє пам'ять для зберігання чотирьох символів (включаючи `'\0'`) і поміщає адресу початкового символу в `st`.

Стандартна бібліотека C пропонує набір функцій для обробки рядків з завершальним нульовим символом. Наприклад,

```
// Повертає довжину рядка (без '\0'):
int strlen(const char *s);
// Порівнює два рядки і повертає від'ємне значення
// якщо s1 менше ніж s2, нуль, якщо вони однакові,
// і додатне значення в іншому випадку:
int strcmp(const char *s1, const char *s2);
// копіює s2 в s1:
strcpy(char *s1, const char *s2);
```

Функція `strcmp()` реалізує порівняння рядків за абеткою. Для використання наведених функцій слід підключити заголовний файл `<cstring>` і простір імен `std`.

2. Структури

Структури є складеними типами, визначеними користувачем. Вони складаються з елементів даних, які можуть мати різні типи. Така група може розглядатися як єдине ціле. Можна визначити новий тип структури даних і пізніше створювати змінні цього типу. Ключове слово `struct` використовується для створення нового типу структури:

```
struct Country
{
    char   name [20];
    double area;
    long   population;
};                      // крапка з комою обов'язкова

Country France;         // France є змінною типу Country
```

Структури, як і інші визначені користувачем типи даних, зазвичай визначаються у глобальній області видимості. Синтаксис мови C++ дозволяє створювати змінні нового типу безпосередньо після його визначення.

```
struct Country
{
    char   name [20];
    double area;
    long   population;
```

```
} France;              // France є змінною типу Country
```

Таке визначення не рекомендується. Більш прийнятним є окреме визначення типів даних і змінних.

Під час створення змінної типу структури можна здійснити її ініціалізацію. Значення елементів ініціалізуються у порядку опису:

```
Country France = {"France", 551695, 64590000 };
```

На відміну від масивів оператор присвоювання виконує поелементне копіювання однієї структури в іншу:

```
Country someCountry;
someCountry = France; // поелементне копіювання
```

Можна отримати доступ до конкретних полів даних з використанням операції «крапка» («.»).

```
cout << someCountry.area;
```

Можна створювати вказівники на структури. Для доступу до елементів структур, на які вказує вказівник, використовується спеціальна операція ->. Наприклад:

```
Country *pCountry;
pCountry->area = 551695;
```

На відміну від масивів, об’єкти типу структури передаються у функції за значенням. Це означає, що функція працює з копіями об’єктів. Якщо необхідно змінити значення елементів структури всередині тіла функції, слід використовувати аргументи типу посилання. Навіть якщо не потрібно змінювати дані всередині функції, можна використовувати посилання на

константні типи, що підвищить продуктивність програми через копіювання адреси замість величезного блоку даних:

```
struct LargeData
{
    // Деяка велика структура
};

void someFunc(const LargeData& data)
{
    ...
};
```

Якщо структура містить низку змінних, які можуть мати лише дуже невелику кількість можливих значень, можна заощадити пам’ять за допомогою бітових полів, вказавши максимальний розмір поля у бітах. Оголошення бітового поля містить ім’я деякого цілого типу, за яким слідує ім’я поля, а потім двокрапка, а потім розмір бітового поля.

```
struct Flags
{
    unsigned int logical : 1; // один біт
    unsigned int tinyInt : 3; // крихітне ціле
};
```

За допомогою бітових полів кілька значень можна зберігати в одному байті. Це не працює, якщо бітові поля чергуються з небітовими полями. Бітові поля можна також використовувати у визначеннях класів.

На відміну від мови програмування С, структури у С++ можуть містити функції-елементи та підтримують інші функції класів. Незважаючи на це структури, як правило, використовуються для групування даних, а не замість класів.

3. Використання зв’язаних списків

Одна із найпоширеніших задач, які розв’язуються програмістами, є подання та обробка послідовностей даних. Більшість задач реального світу може бути розв’язана через використання масивів. Але іноді застосування масивів не є достатнім через такі недоліки:

– додавання нових елементів у кінець масиву не може бути здійснене без додаткових зарезервованих елементів; якщо немає таких елементів, потрібно створити новий масив і скопіювати вміст старого масиву в новий;

– додавання елементів всередину або видалення елементів, як правило, вимагає копіювання багатьох елементів даних.

Інший підхід до подання послідовностей забезпечує так званий зв’язаний список, або ланцюг. Для реалізації зв’язаного списку необхідно створити структуру, яка містить безпосередньо дані й вказівник на інший елемент цього типу:

```
struct Link
{
    Data data;
    Link *next;
};
```

де `Data` – це деякий відомий типі даних. Для додавання нового елемента між існуючими, слід створити цей елемент у динамічній пам’яті, а потім змінити значення вказівників у сусідніх елементах.

Зв’язаний список – це різновид динамічних структур даних. Іншими прикладами таких структур є двонаправлені зв’язані списки, черги, стеки й бінарні дерева.

4. Використання виведення та введення у стилі мови С

Мова С++ дозволяє використовувати функції виведення і введення, успадковані у мови С. Для роботи з цими функціями слід підключити заголовний файл `stdio.h`.

Функцію `printf()` можна використовувати для виведення на консоль. Вона може бути викликана одним або декількома аргументами. Константний рядок може бути виведено на консоль, якщо вказати його як аргумент:

```
printf("Hello"); // так само, як cout << "Hello";
```

Якщо необхідно вивести числа, всередині першого аргументу вказують так звані **символи форматування**. Послідовність форматування починається з символу `%` з подальшим визначенням розміру і безпосереднім символом форматування. Іноді встановлювати розмір поля не треба. Наприклад:

```
int k = 12;
printf("%i", k);
```

Найбільш важливі символи форматування подано у табл. 1.

Таблиця 1 – Найбільш важливі символи форматування

Символи	Дані, які виводяться
`%d` або `%i`	`int`
`%c`	окремий символ
`%e` або `%E`	`float` або `double` у форматі `[-]d.ddd e±dd` або `[-]d.ddd E±dd`
`%f`	`float` або `double` у форматі `[-]ddd.ddd`
`%p`	адреса (вказівник)
`%s`	рядок виведення
`%u`	`unsigned int`
`%x` або `%X`	`int` як шістнадцяткове число
`%%`	символ `%`

Символ «мінус» перед послідовністю форматування обумовлює вирівнювання по лівому краю.

Функція `scanf()` дозволяє зчитувати зі стандартного вхідного потоку. Перший аргумент – це рядок форматування. Він дозволяє, зокрема, вказувати роздільники. Інші аргументи – вказівники на змінні, значення яких повинні бути прочитані. Функція повертає кількість байтів, які читалися. Для читання даних типу `double` слід використовувати `lf` замість `f`. Наприклад:

```
int n;
float x;
double y;
scanf("%d, %f %lf", &n, &x, &y);
printf("%10d ", n);
printf("%10.5f \n", x);
printf("%-10.5f", y);
```

Під час введення перші два числа слід розділити комою.

Недоліком цих функцій у порівнянні з потоками C++ є те, що ці функції є небезпечним з точки зору типів. Їх використання може призвести до численних помилок під час виконання, пов’язаних з невідповідністю типів змінних. Компілятор не може перевірити відповідність під час трансляції.

5. Робота з файлами

Файлові потоки дозволяють здійснювати введення з текстових файлів і виведення у текстові файли. Файлові потоки забезпечують зручність роботи тому, що відкрити файл можна створюючи об’єкт, і файл буде автоматично закритий перед знищенням потоку (через виклик так званого деструктору).

Класи файлових потоків `std::ifstream` і `std::ofstream` визначені у заголовному файлі `fstream`. Файл повинен бути підключений до файлу, перш ніж він може бути використаний. У наведеному нижче прикладі програма зчитує ціле значення з файлу «data.txt» у змінну `k`. Це значення буде записано в інший файл:

```
#include <fstream>
. . .
int k;
std::ifstream inFile("data.txt");
inFile >> k;
std::ofstream outFile("result.txt");
outFile << k;
```

Якщо необхідно перевірити можливість читання даних з файлу, можна перевірити результат методу `eof()`. Можна також перетворити об'єкти потоку в цілі числа і в такий спосіб перевірити стан потоку. Другий підхід є більш ефективним.

Інший підхід – це використання вказівника на файл (`FILE *`) і пари функцій `fscanf / fprintf`. Такий підхід є небезпечним з точки зору типів.

ПРИКЛАДИ ПРОГРАМ

1. Відстань. У наведеному нижче прикладі створено структуру, яка представляє точку на екрані. Програма обчислює відстань між двома точками.

```
#include <iostream>
#include <cmath>

struct Point
{
    int x, y;
};

double sqr(double x)
{
    return x * x;
}

double distance(Point p1, Point p2)
{
    return std::sqrt(sqr(p1.x - p2.x) + sqr(p1.y - p2.y));
}

int main()
{
    Point p1 = {1, 2};
    Point p2 = {4, 6};
    std::cout << distance(p1, p2);
```

```
    return 0;
}
```

Примітка: використання **using namespace** std призводить до помилки компіляції через конфлікт імен.

2. Сума. Текстовий файл з ім'ям «Numbers.txt» містить цілі значення, розділені пробілами, символами табуляції або символами нового рядка. Наприклад:

```
1 22 -3 11    -9 144
1000
1 1 1 -3 5    4
3 55    2
```

Довжина файлу не обмежена. Необхідно створити програму, яка зчитує цілі значення до кінця файлу і обчислює суму цих чисел.

Файл з вихідними даними повинен бути поміщений у робочий каталог проекту. Сирцевий код програми може бути таким:

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    ifstream in("Numbers.txt");
    int x, sum = 0;
    while (in >> x) // читання успішне
    {
        sum += x;
    }
    cout << sum;
    return 0;
}
```

3. Зворотний порядок. Наведена нижче програма читає числа з рухомою комою з текстового файлу і поміщає ці числа у новий файл у зворотному порядку. Дані вводяться до кінця файлу. Під час завантаження даних створюється тимчасовий зв'язаний список. Структура Link

використовується для подання конкретних елементів списку. Завантажені дані переписуються у новий масив, а потім тимчасові структури даних знищуються.

```
#include <iostream>
#include <fstream>
using namespace std;

struct Link
{
    double data;
    Link *next;
};

double *readFromFile(char *fileName, int &count)
{
    Link *first = 0;
    Link *last = 0;
    Link *link;
    ifstream in(filename);
    double d;
    count = 0;
    while (in >> d)
    {
        count++;
        link = new Link;
        link->data = d;
        link->next = 0;
        if (last == 0)
        {
            first = last = link;
        }
        else
        {
            last->next = link;
        }
        last = link;
    }
    double *arr = new double[count];
    link = first;
    for (int i = 0; i < count; i++)
    {
        arr[i] = link->data;
        link = link->next;
    }
    while (first)
    {
        link = first;
```

```
        first = first->next;
        delete link;
    }
    return arr;
}

void outToFile(char *filename, double *arr, int count)
{
    ofstream out(filename);
    for (int i = count - 1; i >= 0; i--)
    {
        out << arr[i] << " ";
    }
}

int main()
{
    int count = 0;
    double *arr = readFromFile("data.txt", count);
    outToFile("results.txt", arr, count);
    delete [] arr;
    return 0;
}
```

ВПРАВИ ДЛЯ КОНТРОЛЮ

Завдання 1. Визначити структуру для подання двох цілих чисел, а потім створити і викликати функцію, яка отримує як аргумент структуру створеного типу і обчислює добуток елементів структури.

Завдання 2. Написати програму, яка читає цілі до значення 0 і обчислює добуток цих чисел без останнього нульового значення.

Завдання 3. Написати програму, яка читає цілі числа до кінця файлу і обчислює добуток ненульових значень.

Завдання 4. Написати програму, яка визначає масив чисел і записує у текстовий файл суми елементів з непарними індексами.

ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ

1. Точки у тривимірному просторі

Написати програму, яка обчислює відстань між двома точками у тривимірному просторі. Вивести результат за допомогою функції `printf()`.

2. Середнє арифметичне

Написати програму, яка зчитує значення з рухомою крапкою з текстового файлу до кінця файлу і обчислює середнє арифметичне цих значень. Вивести результат за допомогою функції `printf()`.

3. Індивідуальне завдання

Написати програму, яка забезпечує файлове введення та виведення і включає індивідуальне завдання попередньої лабораторної роботи. Слід реалізувати такі дії:

– визначення константи (`n`), яка визначає кількість стовпців двовимірного масиву;

– відкриття файлу для читання (файл повинен бути підготовлений за допомогою текстового редактора);

– читання цілих чисел до кінця файлу і зберігання їх у зв’язаному списку;

– створення двовимірного масиву в динамічній пам’яті; кількість рядків повинна бути обчислена на основі кількості зчитаних з файлу значень та визначених стовпців;

– заповнення двовимірного масиву рядок за рядком; відсутні елементи останнього рядка повинні бути заповнені нулями;

– видалення елементів зв’язаного списку з динамічної пам’яті;

– реалізація попереднього індивідуального завдання;

– зберігання результатів у новому файлі;

– видалення масивів операцією `delete`.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. У чому є специфіка символьних масивів?
2. Що таке рядок, який закінчується нулем (`null-terminated string`)?
3. Визначте поняття структури.
4. У чому різниця між структурами і масивами?
5. Чому слід розташувати крапку з комою після дужки, яка закриває тіло структури?
6. Як надіслати структури до функцій?
7. Що таке бітові поля?
8. У яких випадках бітові поля підвищують ефективність програми?
9. Чи може структура містити функції-елементи?
10. Що таке динамічні структури даних?
11. Які переваги зв’язаних списків у порівнянні з масивами?
12. Які недоліки зв’язаних списків у порівнянні з масивами?
13. Які функції мови С використовують для виведення і введення?
14. Які є недоліки використання функцій `printf()` і `scanf()`?

Чому явне закриття файлів у С++ не є обов’язковим?

Рекомендована література

1. Bjarne Stroustrup. The C++ Programming Language. Third Edition / Bjarne Stroustrup. – Addison-Wesley, 1997.

2. Stanley B. Lippman C++ Primer. Third Edition / Stanley B. Lippman, Josee Lajoie. – Addison-Wesley, 1988.

3. Deitel H. M. C++. How to Program. Third Edition / H. M. Deitel, P. J. Deitel. – Prentice Hall, 2001.

4. Голуб Б. М. C#. Концепція та синтаксис / Б. М. Голуб. – Львів: Видавничий центр ЛНУ ім. Івана Франка, 2006. – 136 с.

5. Грицюк Ю. І. Програмування мовою С++: навч. посібник / Ю. І. Грицюк, Т. Є. Рак. – Львів : Вид-во Львівського ДУ БЖД, 2011. – 292 с.

Internet-джерела

1. The C++ Programming Language (Bjarne Stroustrup's homepage) // http://www2.research.att.com/~bs/C++.html

2. ISO/IEC 14882:2003 Programming languages - C++ (International Stamdard) // http://cs.nyu.edu/courses/summer12/CSCI-GA.2110-001/downloads/C++%20Standard%202003.pdf

3. The C++ Resources Network // http://www.cplusplus.com/

4. The C++ Tutorial // http://www.learncpp.com/

5. C++ – Вікіпідручник // http://uk.wikibooks.org/wiki/C++

6. C++ – Вікіпіедія // http://uk.wikipedia.org/wiki/C++

7. Корх О. Основи мови програмування С++ // http://korkholeh.googlepages.com/cppfund.pdf

Навчальне видання

Методичні вказівки

до виконання лабораторної роботи 5

за темою «Робота зі структурами та файлами»

з курсу «Алгоритмізація та програмування. Частина 1»

для студентів спеціальності

122 «Комп’ютерні науки»

Укладачі:

ІВАНОВ Лев Вадимович

БІЛОВА Марія Олексіївна

Відповідальний за випуск М. Д. Годлевський

Роботу до видання рекомендував О. В.Горілий

План 2018 р., поз. 314

Підписано до друку 28.05.2019. Гарнітура Times New Roman.

Ум. друк, арк. 0,8.

Видавничий центр НТУ «ХПІ»,

вул. Кирпичова, 2, м.Харків-2, 61002

Свідоцтво про державну реєстрацію ДК № 3478 від 21.08.2017 р.

Самостійне електронне видання